1993

Multidatabase Atomic Commitment Protocols: A Taxonomy and Unified Approach

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

James G. Mullen

Report Number:
93-018

MULTIDATABASE ATOMIC COMMITMENT PROTOCOLS: A TAXONOMY AND UNIFIED APPROACH

Ahmed K. Elmagarmid
James G. Mullen

CSD-TR-93-018
March 1993

Multidatabase Atomic Commitment Protocols: A Taxonomy and Unified Approach*

Ahmed K. Elmagarmid
James G. Mullen
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

Atomic commitment deals with the problem of ensuring that either all or none of the subtransactions of a multi-system transaction are made permanent. Atomic commitment is made difficult in multidatabase systems due to component system autonomy, which not only makes the implementation of correct atomic commitment protocols more difficult, but also can decrease the acceptability of blocking. Various levels of support for commitment may be provided by the different component systems in a multidatabase system, and this paper addresses the issue of providing atomic commitment in such an environment. A taxonomy of multidatabase atomic commitment methods is provided. This taxonomy is used to develop a subtransaction taxonomy, which is used in conjunction with an execution model to define the types of multidatabase transactions that are committable, and to construct a unified multidatabase atomic commitment protocol.

Index Terms: atomic commitment protocols, federated database systems, heterogeneous distributed database systems, multidatabase systems, transaction management

1 Introduction

Multidatabase systems combine autonomous and heterogeneous component (or local) database systems into a global database system. Transactions represent logical units of work in database systems, and in multidatabase systems there are two types of transactions: global transactions and local transaction. Global transactions are divided into subtransactions, with one subtransaction per local (or component) system that the global transaction accesses. Local transactions execute at a single local database system. Global transactions are submitted to the multidatabase system, and local transactions are submitted directly to local database systems. A conceptual view of a multidatabase system is shown in Figure 1. A global transaction G_i, and its decomposition into subtransactions $G_{i,1}$, $G_{i,2}$, ..., $G_{i,n}$ is shown. Also, a local transaction $L_{j,1}$ that executes at $LDBS_1$, and a local transaction $L_{k,n}$ that executes at $LDBS_n$, are shown.

One of the most difficult problems with implementing reliable transaction management in multidatabase systems is the problem of atomic commitment of global transactions. That is, ensuring that if any of the effects of a global transaction are executed, then all of the effects will be executed. In fact, in [19] it has been shown that it is *impossible* to do in general without violating local autonomy. The two phase commit

*This research funded by a PYI Award from NSF under grant IRI-8857952, a grant from the Software Engineering Research Center at Purdue University (a National Science Foundation Industry/University Cooperative Research Center — NSF Grant No. ECD-8913133), A Graduate Student Researchers Program Grant from NASA, and grants from the AT&T Foundation, Tektronix, Mobil Oil and Bell Northern Research.

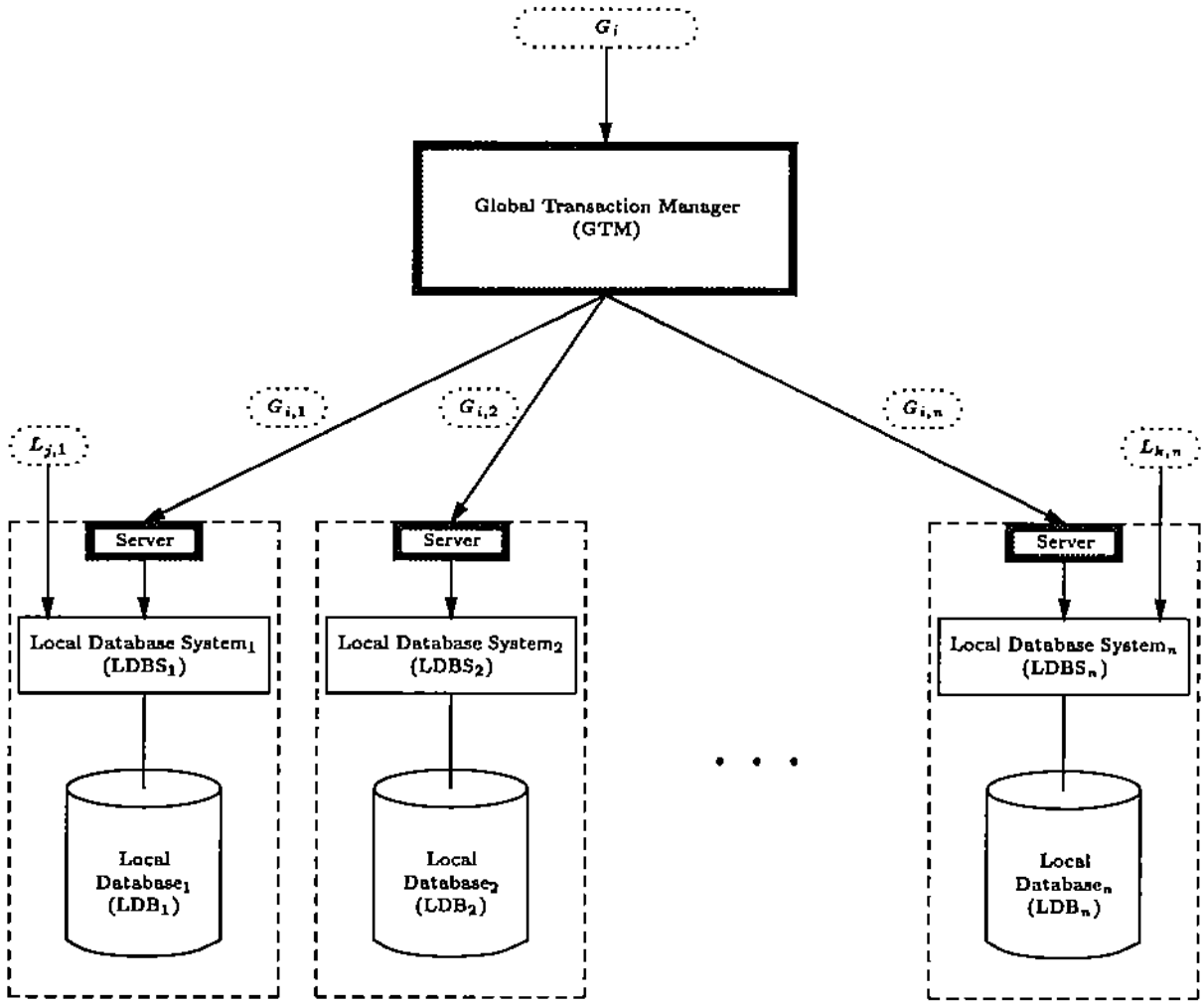

Figure 1: Conceptual Multidatabase Architecture.

protocol (2PC) [14, 11] has been used for tightly coupled distributed database systems. However, there are at least two problems with applying 2PC to the multidatabase case. First, current database systems do not generally provide the (visible) prepare-to-commit state which is necessary to implement 2PC, and if even one component system at which a transaction executes does not support the prepare-to-commit state, it will not be possible to use the 2PC algorithm. Second, even if database systems used in the future generally do provide the prepare-to-commit state, 2PC can severely violate local execution autonomy. Blocking can often occur with 2PC, so it becomes possible for a remote system to, in effect, lock another system's data for indefinite periods of time.

Various approaches to providing atomic commitment in multidatabase systems have been proposed that do not require a visible prepare-to-commit state. For example, compensation approaches have been proposed where atomicity is maintained by semantically undoing committed subtransactions when the global transaction aborts. And, several "redo" approaches have been proposed where atomicity is maintained by re-executing failed subtransactions. In general, these approaches assume that all the component systems in the multidatabase can support the proposed protocol. However, it is possible that different component systems will provide support for different commitment protocols. And, given the autonomous and heterogeneous nature of multidatabase component systems, it is not only possible, but probable. Therefore, one needs an approach to atomic commitment that can integrate component systems that have various levels of

support for atomic commitment.

In this paper we will present a unified approach to multidatabase atomic commitment that will work for multidatabase systems consisting of component systems with various levels of commitment support. This paper is organized as follows. Section 2 provides background information. Section 3 presents a taxonomy of multidatabase atomic commitment protocols. Section 4 presents a taxonomy of multidatabase subtransactions that is based on the commitment approach of the component system where the subtransaction executes (based on our commitment protocol taxonomy), and the specific semantics of the subtransaction. A state-transition execution model for multidatabase global transactions is developed, and is used in conjunction with the subtransaction taxonomy to classify which global transactions are, and are not, committable. Section 5 presents a unified commitment approach based on the subtransaction taxonomy and execution model of Section 4. Section 6 presents some related work. Finally, Section 7 presents our conclusions.

2 Background

2.1 Multidatabase Atomic Commit

In multidatabase systems there are two types of transactions: local and global. Local transactions execute at only one component system and are submitted directly to that component system. Global transactions may execute at multiple components systems in the multidatabase, and are configured to have one subtransaction per component at which they execute. Multidatabase atomic commitment is concerned with the correct execution of the global transactions, and the goal is to ensure that either all or none of the subtransactions of a global transaction commit, i.e. that there are no partial commitments of a global transaction. The component systems will ensure that the individual subtransactions are either fully executed and committed, or aborted with no effect.

2.2 Correctness of Atomic Commitment

We will define an atomic commitment protocol to be correct if all of the following conditions are maintained:

1. The protocol must terminate given a large enough (yet finite) amount of failure-free time, where failure-free does *not* consider *integrity constraint transaction failures.* That is, failures caused by the state of the data in the database. For example, if a withdraw subtransaction failed because the account from which it was withdrawing didn't have enough money.

2. Upon termination, the protocol must either have committed all of the subtransactions or none of them.

3. Once the protocol terminates with commitment, the global transaction cannot be aborted.

4. If each subtransaction is able to commit at its component system, then the protocol must commit the transaction globally.

5. The values read by any committed subtransaction from another subtransaction belonging to the same transaction, will be equivalent to the values read by the committed version of the subtransaction from which it is reading.

The above definition is constructed so that it will include compensation and redo commitment approaches, as well as the more traditional two phase commitment approach. For example, we say that once the global transaction has committed, it cannot be aborted, but we do not place such a restriction on individual subtransactions, therefore allowing compensation to be considered as an atomic commitment protocol.

Also, we want to allow the re-execution of failed subtransactions as a method of atomic commitment. One of the problems that can occur with this approach is that a subtransaction reads from another subtransaction,

belonging to the same transaction, that ends up eventually committing, but commits reading different values than the execution that was read. In effect, a non-recoverable execution occurs. To prevent this problem, the last condition is proposed. Note that according to the condition, a subtransaction has to read from a subtransaction equivalent to the execution that commits, not the actual execution that commits.

2.3 Limitations of Multidatabase Atomic Commitment

In [19] we showed conditions where atomic commitment is not possible in multidatabase systems. If one only assumes that component database systems support the transaction abstraction (and not how they support it), then there is no atomic commitment protocol that will work in general for multidatabase systems. And, it turns out that this is true even when system and communication failures cannot occur. In addition, even if one assumes that all component systems use strict two-phase locking for the concurrency control method, atomic commitment still is not possible. This is true if even a single component system failure can occur.

As a result of atomic commitment being impossible in general without the violation of autonomy, one must employee one of two basic strategies in order to implement atomic commitment in a multidatabase system. One must either violate the autonomy of the component systems, or one must restrict the class of transactions allowed in the system. And, of course, one may try a combination of the two strategies.

3 A Taxonomy of Multidatabase Atomic Commitment Protocols

Our taxonomy of multidatabase atomic commitment protocols is structured as a two-level hierarchy. The first level of the hierarchy is based in the basic approach (or strategy) used:

1. **Prepare Approach.** One attempts to execute all subtransactions up to a prepare-to-commit state (just before the commit point) where the subtransaction is guaranteed to be able to either commit or abort depending on the protocol coordinator's decision. A consensus must be reached that all component systems can commit before deciding to commit. This is the approach that is used with two phase commitment.

2. **Redo Approach.** In this approach, one attempts to guarantee that the execution of each subtransaction will eventually succeed. If this guarantee can be made, then one re-executes failed subtransactions (if any) until all subtransactions complete.

3. **Undo Approach.** In this approach, subtransactions may be undone. So, the subtransactions can be committed independently. If a subtransaction should fail after other subtransactions belonging to the same transaction have already committed, the committed subtransactions can simply be undone.

The second level of the hierarchy is based on the specific implementation method used to solve the problems inherent in the basic approach used. These problems, and their solutions, will be covered in the rest of this section.

3.1 Prepare Approach

3.1.1 Problems

There is basically only one problem with implementing a prepare approach, and that is implementing the visible prepare-to-commit state. Currently, the vast majority of database systems available do not provide a prepare-to-commit state that is externally visible.

3.1.2 Solutions

Modify Component System. One solution is to modify the component system so as to add a visible prepare-to-commit state. This will often not be a reasonable solution as many times an organization may only have the executable version of their database system. Even if the source code version exists, the organization may not have the resources to modify the system.

There are approaches to simulate a prepare-to-commit state for systems that do not have one, however, we know of no such approaches that would not come under the redo or undo approach options. That is, it seems that a prepare-to-commit state is always simulated by using an undo or redo approach.

3.2 Redo Approach

3.2.1 Problems

Implementing the redo approach has two basic problems:

1. Avoiding intra-transaction non-recoverability. This problem results from the fact that a subtransaction may commit having read values from another subtransaction (belonging to the same global transaction) that has not committed. The problem can occur if the uncommitted transaction ends up failing and is redone, but reads different values on the retry attempt when it commits.

2. Avoiding indefinite retry failure. This problem can occur because the state of the database may change in such a way that the re-execution of a subtransaction will abort indefinitely. Suppose, for example, a subtransaction that withdraws $100 from an account aborts, and then the account drops below $100 due to some other transaction. Attempts to retry the subtransaction could abort indefinitely.

3.2.2 Solutions

All solutions to the redo approach use either a restriction on transactions, violation of local autonomy, or both. The methods listed below are grouped by the mechanism used to implement them.

Transaction Class Restriction. One solution is to restrict the class of transactions allowed. For example, to solve the problem of intra-transaction non-recoverability, one may disallow global transactions that have cyclical "read-from" dependencies. That is, only allow transactions whose subtransactions can be committed in some sequential order without having a subtransaction having read values from an uncommitted subtransaction. To solve the problem of indefinite retry failure one can allow only global transactions that have at most one subtransaction that is susceptible to integrity constraint transaction failures. Furthermore, one must be able to execute this subtransaction first. So, unfortunately, if one simply restricts the transaction class to implement the redo approach, the resulting class is fairly restrictive. This approach is examined in detail in [7]. Also, in [12], a retry approach is presented for atomic commitment that restricts the class of transactions allowed. Essentially, only transactions whose subtransactions cannot have integrity constraint based transaction failures are allowed.

Data Partitioning Strategy. Another solution is to partition the data into globally and locally updatable data. That way, as long as global transactions are prevented from executing until current transactions executing that access their data are complete, the subtransaction can be guaranteed to commit. This approach is used in [3], [4], [17], and [24].

Local Transaction Elimination/Rerouting. Another solution is to require local transactions to go through the global transaction manager, or at least a server of that manager. If this solution is used then the global transaction manager gains knowledge of, and control over, the local transactions and can prevent the local transactions from accessing data of a global transaction that needs to be redone. Such an approach

is presented in [20] and in [22]. There are two problems with this approach. First, there is a performance degradation for local transactions due to an extra software layer. Second, it may require substantial effort to duplicate the component system interface. For example, if a complex graphical user-interface was provided, it would either have to be duplicated, or users would have to be retrained.

Exclusive Component System Access. In [1] and [10] methods are presented that assume the GTM can exclusively access the LDBMSs after they fail. Local transactions are prevented from executing until the GTM can redo its global subtransactions at the LDBMS. Therefore, this method violates local system autonomy.

Local Transaction Modification and Data Item Addition. While modifying the component system is usually practically impossible, modifying local transaction programs may be possible. In the method proposed in [9] (called reservation commitment), local transactions that update data that may be accessed by global transactions are modified so that commitment can be performed. In addition, additional data items may need to be added to the component database.

The reservation solution in [9] uses a reservation phase that attempts to ensure that a subtransaction can eventually be committed by re-executing it. For example, if a subtransaction needed to withdraw $100 from an account, the reservation phase might ensure that there is $100 in the account, and that no future transaction brings the account below $100 until the subtransaction can complete. A *lower_limit* data item could be added, and the reservation would set it at $100. Local transactions would be modified to abort if they brought the account below the lower limit.

In the above example, the semantics of the data are used to reduce the level of blocking. Any transaction that does not make the account go below $100 is free to proceed. However, a general approach that would completely lock the data from updates could be used. Read-only transactions would still be able to proceed.

3.3 Undo Approach

3.3.1 Problem

The basic problem with the undo approach is the inability to undo a subtransaction and still maintain correctness. A subtransaction may be non-compensatable for many reasons. If the subtransaction is not commutative with other transactions that may be issued at the system, then the subtransaction cannot be compensated [13]. Basically, if the order of execution is important (i.e. the transactions are non-commutative), then a non-recoverable execution could be generated, since other transactions may access the data in between a subtransaction and its compensation and see the effects of the subtransaction that will be compensated. Also, if the subtransaction's effects correspond to some physical event that cannot be compensated (e.g. launching a missile, or dispencing money from an automatic teller machine) then the subtransaction cannot be compensated.

3.3.2 Solutions

All solutions to the undo approach use either a restriction on transactions, violation of local autonomy, or both. The methods listed below are grouped by the mechanism used to implement them, and correspond to categories used for the redo approach.

Transaction Class Restriction. One solution is to restrict the class of transactions allowed. One may allow only subtransactions that are commutative with other subtransactions and local transactions. A problem with this, however, is that it would seem to require knowledge of local transactions, and notification of the global transaction manager, should the local transactions be modified, or additional local transactions added. This would seem to violate autonomy.

Data Partitioning Strategy. Another solution is to partition the data into globally and locally updatable data. That way, as long as global transactions are prevented from executing until current transactions executing that access their data are complete, the subtransaction can be guaranteed to commit. This assumes that one could save the original data values, and rewrite them, should the global decision be to abort. One does not need to have commutative operations, since operations could be completely blocked on data accessed by a subtransaction that might later be compensated. On the other hand, one loses the non-blocking advantage of compensation by using this approach. In addition, this approach will still not solve the problem of subtransactions that cause an irreversible physical phenomenon.

We do not know of anyone who has proposed this approach so far, and one possible problem with it could be the inability to restore the previous state to the component system. Suppose, for example, that the global database system is only allowed to read the balance of, and to deposit money into, a component system's account. Certainly, the global system could get the value before a deposit would be performed and store it in stable storage, but there would be no way to update the account to its original value once the deposit was made. So, it could be that because of autonomy reasons, a component does not allow arbitrary updates to its data, and the global system has no way of restoring previous values to compensate a subtransaction.

Local Transaction Elimination/Rerouting. Another solution is to require local transactions to go through the global transaction manager, or at least a server of that manager. If this solution is used, then the global transaction manager gains knowledge of, and control over the local transactions and can prevent the local transactions from accessing data of a global transaction that needs to be undone. Such an approach is presented in [20] and [21]. This approach would have the same problems mentioned in the analogous redo approach, and the non-blocking advantage of compensation would be lost.

Local Transaction Modification and Data Item Addition. Another solution is to modify local transactions so that compensation can be performed. This approach is discussed in [9]. In the approach discussed, a reservation phase is used for a compensation that guarantees that the compensation will eventually commit. For example, suppose the compensation for a transaction that deposits $100 into an account, it to withdraw $100 dollars from the account. This compensation could abort indefinitely if the account is not allowed to go below $0, and some transaction makes the account go below $100 before the compensation can be performed. The reservation phase would ensure that no transaction would make the account go below $100 until either the compensation could be performed, or the entire global transaction had been committed.

4 A Taxonomy of Subtransactions

In the previous section, several different commitment protocols were presented. These protocols generally assume that each component has the capabilities to support the commitment approach used by the protocol. However, it may well be that different component systems can support different commitment methods. This section develops theory for supporting a unified commitment approach that can combine various component system capabilities.

In this section we define a taxonomy of subtransactions based on properties that are important relative to performing atomic commitment. The classification of a subtransaction will be dependent on three factors:

1. The commit protocol(s) supported by the component system at which it executes.
2. The subtransaction's semantics.
3. The operational interface provided by the component system. Specifically, is an explicit commit operator provided or not.

Clearly, the commit protocols supported by the component system will affect how subtransactions that execute at it are classified. Undo approaches would generally support the notion of compensation, and redo

approaches would generally use a reservation concept. That is, they would somehow guarantee that the re-execution of the subtransaction would eventually succeed.

The specific semantics of the subtransaction would also affect the commitment protocol. For example, a read-only subtransaction at a component system with no compensation method could be considered as implicitly compensatable, and probably implicitly reservable.

Another important aspect of subtransactions is whether or not they have an explicit commit operator [22]. That is, is it possible to run the subtransaction up to its commit point without committing it, or does the entire subtransaction have to be sent at once. Clearly, if a system supports a visible prepare-to-commit state, it must have an explicit commit, however, in the reservation or compensation case either option is possible. It is important to consider this case, because many database systems, particularly older ones, do not provide an explicit commit operator. In addition, it is often desirable to integrate non-database systems into a multidatabase system. These systems will generally not provide an explicit commit operator, and often the only way the atomicity of subtransactions to such systems can be guaranteed is by submitting a single command, or, perhaps, by submitting a group of commands together.

From the viewpoint of supporting commitment, this issue is important because it will affect the order of read dependencies that is allowed. For example, if a subtransaction with no global commitment method does not have an explicit commit operator, then it may be impossible for a preparable subtransaction to read from it. If it does have an explicit commit operator, then it may be possible.

Our taxonomy of subtransactions is as follows:

1. **Implicitly Compensatable (IC)** — a subtransaction that can be considered as compensated after it commits, without the need for an explicit compensation, such as a read-only subtransaction.

2. **Compensatable (C)** — a subtransaction that can be undone after it is committed, but before the global transaction commits, by executing an explicit compensation at the subtransaction's system. Generally, systems that support an undo approach will support this type of subtransaction.

3. **Reservable Compensatable (RC)** — a subtransaction that can be undone after it is committed, but before the global transaction commits, by executing an explicit compensation at the subtransaction's system. The compensation step in this case requires a reservation step that must be successfully executed before the subtransaction is executed. The undo approach discussed in [9], which uses local transaction modification, can support such subtransactions.

4. **Preparable (P)** — a subtransaction that can be prepared. That is, executed up to its commit point, and guaranteed to be commitable or abortable based on what the global coordinator decides. Component systems that provide a visible prepare-to-commit state, or that are modified to provide a prepare-to-commit state will support such subtransactions.

5. **Implicitly Reservable (IR)** — a subtransaction that is guaranteed to commit if redone, but does not require any explicit reservation step. The values read by the subtransaction are not guaranteed to be the same for different executions in this case. A subtransaction in this class could result from the semantics of the subtransaction, or from the method of commitment supported at the component system. An example of such subtransaction semantics could be a subtransaction that withdraws all the money in an account, regardless of how much is in it. An example of a commitment approach supporting this type of subtransactions is the data partioning redo approach. Because of the data partioning (and use of stable storage) the subtransaction can be guaranteed to be redoable.

6. **Value Preserving Implicitly Reservable (VPIR)** — a subtransaction that is guaranteed to commit if redone, but does not require any explicit reservation step. The values read by the subtransaction are guaranteed to be the same for different executions in this case. The data partioning redo approach will support subtransactions in this class.

7. **Reservable (R)** — a subtransaction that is guaranteed to commit if redone, but requires an explicit reservation step. The values read by the subtransaction are not guaranteed to be the same for different executions in this case. The redo method in [9], that uses local transactions modification, can support subtransaction in this class.

8. **Value Preserving Reservable (VPR)** — a subtransaction that is guaranteed to commit if redone, but requires an explicit reservation step. The values read by the subtransaction are guaranteed to be the same for different executions in this case. An example could be a subtransaction that withdraws $100 from an account whose reservation step guarantees no other transactions will modify the account balance until the subtransaction completes. The redo method in [9], that uses local transactions modification, can support subtransaction in this class.

9. **Non-Compensatable-Preparable-Reservable (NCPR)** — a subtransaction that is not compensatable or reservable, either implicitly or explicitly, and is not preparable either.

We will use the abbreviation of each category to refer to the set of subtransactions that are in the category. For example, we will use IR to refer to the set of all subtransactions that are implicitly reservable. Furthermore, we will use XC to refer to the set of subtransactions that have an explicit commit operator, and NXC to refer to the set of subtransactions that do not have an explicit commit operator.

We will use RES to refer to the set of subtransactions that are reservable (i.e. in $R \cup IR \cup VPR \cup VPIR$), and COMP to refer to the set of subtransactions that are compensatable (i.e. in $IC \cup C \cup RC$).

Not all subtransaction sets are disjoint. However, a subtransaction in the $NCPR$ set cannot be a member of any other set, and an implicit reservation must be either value-preserving or not. Also, a subtransaction must either have an explicit commit operator, or not have one, and a subtransaction in P will necessarily have an explicit commit operator. Some class combinations might not make much sense, but could occur. It might not make much sense to have an implicit and explicit compensation approach, but it could make sense to have a reservation and prepare-to-commit approach for the same subtransaction. Or, one might well have a value preserving and non-value preserving reservation method for the same subtransaction, where the value-preserving one allows more possible "read-from" dependencies, but may have a higher level of blocking.

We state the subtransaction class restrictions more formally below. For all subtransactions s:

- $s \in \{XC \cup NXC\}$
- $s \in XC \Leftrightarrow s \notin NXC$
- $s \in \{NCPR \cup IC \cup C \cup RC \cup P \cup IR \cup VPIR \cup R \cup VPR\}$
- $s \in NCPR \Leftrightarrow s \notin \{IC \cup C \cup RC \cup P \cup IR \cup VPIR \cup R \cup VPR\}$
- $s \in IR \Rightarrow s \notin VPIR$
- $s \in VPIR \Rightarrow s \notin IR$
- $s \in P \Rightarrow s \in XC$

Not only may a subtransaction belong to multiple classes, which implies that it can use multiple commitment methods, it may actually use multiple commitment methods within the same execution. For example, if a subtransaction is reservable and compensatable, one may first execute the reservation (to ensure commitment), and then the compensation later (to undo the subtransaction).

4.1 Transaction Execution Model

In this section we will discuss our model of the execution status of a transaction. This model will be important in showing what types of global transaction are, and are not, committable.

Our model represents the status of a transaction as a vector of subtransactions states. Each subtransaction must be in exactly one of the following states:

- Uncommitted
  - Non-recoverable
    - Values Unknown (U)
    - Values Known (U_v)
  - Recoverable
    - Values Unknown (U_r)
    - Values Known ($U_{r,v}$)
- Uncommitted with Commitment Guaranteed
  - Non-recoverable
    - Values Unknown (U^c)
    - Values Known (U_v^c)
  - Recoverable
    - Values Unknown (U_r^c)
    - Values Known ($U_{r,v}^c$)
- Uncommitted with Values and Commitment Guaranteed
  - Non-recoverable
    - Values Unknown ($U^{c,v}$)
    - Values Known ($U_v^{c,v}$)
  - Recoverable
    - Values Unknown ($U_r^{c,v}$)
    - Values Known ($U_{r,v}^{c,v}$)
- Committed
  - Non-recoverable (C)
  - Recoverable (C_r)
- Aborted (A)

So, for example, the status of an aborted transaction with three subtransactions would be

$$[A, A, A]$$

We refer to a subtransaction as having its values known if it knows the values it will read from the database it accesses. For subtransactions that are re-executed, it only implies that the subtransaction knows the values of its current or last execution, and not necessarily the values it will read from its component database for the execution that actually commits.

Recoverable has a slightly different meaning here than in the standard context. First of all, we are refering to intra-transactional recoverability. Second, the subtransactions that are "read from" do not have to be committed, they can also be uncommitted subtransactions whose values are known, and whose values and commitment are guaranteed. We will use $RF(s)$ to refer to the subtransactions (belonging to the same transaction) from which subtransaction s reads. By "reads" we will mean any subtransactions whose values

affect s, so we will consider indirect, as well as direct, reads. Therefore, our read-from definition is, in effect, the transitive closure of direct subtransaction reads. So, if $u \in RF(t)$ and $t \in RF(s)$, then $u \in RF(s)$. Specifically, a subtransaction can move to a recoverable state if all the subtransactions from which it reads are in any of the following execution states:

- $U_v^{c,v}$
- $U_{r,v}^{c,v}$
- C
- C_r

If a subtransaction's commitment is guaranteed, it implies that the subtransaction will be guaranteed to commit if it is re-executed when it fails, and given enough failure-free time. If a subtransaction's values are guaranteed, this means that once the values that it reads from the component database system it accesses are known, they are guaranteed to be the same values that will be read if the subtransaction commits.

A transaction is committed correctly when all subtransactions are in state C_r. A subtransaction is aborted when all subtransactions are in state A. We will assume that all subtransactions start in state U, although certain subtransaction (e.g. implicitly reservable ones) can immediately move to another state.

We will formalize our subtransaction commitment taxonomy definitions by showing how the subtransaction's commitment type affects the allowable transitions that the subtransaction's execution status can make. The allowable state transitions based on the subtransaction types are shown in Tables 1 and 2.

The transitions in Tables 1 and 2 are essentially complete, however a few comments are in order. First, we do not include certain transitions that are combinations of transitions that are shown. For example, one could argue that a subtransaction that is implicitly reservable and reads from no other subtransaction could make the transition $U \rightarrow U_r^c$. However, including such transitions does not affect the correctness of our model, so we do not include them for the sake of brevity. Second, we consider a subtransaction that is redone to stay in its (uncommitted) state before its execution, even if the execution fails. For example, if a subtransaction that is redone is in state $U_{r,v}^c$ before its execution, it will remain in state $U_{r,v}^c$ even after it is executed and fails; it does *not* move to state A. Third, it is possible that a subtransaction that is considered recoverable will have a subtransaction from which it reads abort. One might argue that the subtransaction should now be made non-recoverable, however this point is not important since a subtransaction cannot make a transition from the aborted state. So, once one subtransaction moves to the aborted state A, all subtransactions must move to the aborted state for atomicity to be maintained. If they cannot, then the fact that a subtransaction is considered to be in state C_r, when it should be in state C, is of no consequence.

Each transition, where the condition is dependent on the execution of the subtransaction, can be made given a sufficient (yet finite) amount of failure-free time. Other transitions can be made immediately (assuming the condition holds). Transitions that do not have a condition can be made automatically.

4.2 Commitable Global Transactions

In this section we present theorems that describe the types of global transactions that are and are not commitable.

Theorem 4.1 *A global transaction T will not be committable if it contains more than one subtransaction that is not compensatable, preparable, or recoverable (i.e. $t, s \in T$, $t, s \in NCPR$, and $s \neq t$).*

Proof: The transaction will start in the following state:

$$[..., U, ..., U, ...]$$

where the two U's represent the state of s and t. One can see from the transition table that the subtransactions have no possible way to move to a state where the commitment is guaranteed (U^c, U_v^c, U_r^c, etc.).

Transition	Condition (for subtransaction s)
$U \rightarrow U_r$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U \rightarrow U_v$	if $s \in XC$, $s \notin P$, and execution up to commit point successful
$U \rightarrow U^c$	if $s \in IR$ or ($s \in R$ and reservation is successful)
$U \rightarrow U^{c,v}$	if $s \in VPIR$ or ($s \in VPR$ and reservation is successful)
$U \rightarrow U_v^{c,v}$	if $s \in P$ and execution up to commit point successful
$U \rightarrow C$	$s \in NXC$, execution successful
$U \rightarrow A$	$s \in NXC$, execution not successful
$U_v \rightarrow U_{r,v}$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U_v \rightarrow C$	execution successful
$U_v \rightarrow A$	execution unsuccessful
$U_r \rightarrow U_{r,v}$	if $s \in XC$, $s \notin P$, and execution up to commit point successful
$U_r \rightarrow U_{r,v}^{c,v}$	if $s \in P$, and execution up to commit point successful
$U_r \rightarrow C_r$	execution successful
$U_r \rightarrow A$	execution unsuccessful
$U_{r,v} \rightarrow C_r$	execution successful
$U_{r,v} \rightarrow A$	execution unsuccessful
$U^c \rightarrow U_r^c$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U^c \rightarrow U_v^c$	if $s \in XC$, $s \notin P$, and execution up to commit point successful
$U^c \rightarrow U_v^{c,v}$	if $s \in P$, and execution up to commit point successful
$U^c \rightarrow C$	
$U^c \rightarrow A$	
$U_v^c \rightarrow U_{r,v}^c$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U_v^c \rightarrow C$	
$U_v^c \rightarrow A$	
$U_r^c \rightarrow U_{r,v}^c$	if $s \in XC$, $s \notin P$, and execution up to commit point successful
$U_r^c \rightarrow U_{r,v}^{c,v}$	if $s \in P$ and execution up to commit point successful
$U_r^c \rightarrow C_r$	
$U_r^c \rightarrow A$	
$U_{r,v}^c \rightarrow C_r$	
$U_{r,v}^c \rightarrow A$	

Table 1: Subtransaction State Transitions I

Transition	Condition (for subtransaction s)
$U^{c,v} \rightarrow U_r^{c,v}$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U^{c,v} \rightarrow U_v^{c,v}$	if $s \in XC$ and execution up to commit point successful
$U^{c,v} \rightarrow C$	
$U^{c,v} \rightarrow A$	
$U_r^{c,v} \rightarrow U_{r,v}^{c,v}$	if $s \in XC$ and execution up to commit point successful
$U_r^{c,v} \rightarrow C_r$	
$U_r^{c,v} \rightarrow A$	
$U_v^{c,v} \rightarrow U_{r,v}^{c,v}$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$U_v^{c,v} \rightarrow C$	
$U_v^{c,v} \rightarrow A$	
$U_{r,v}^{c,v} \rightarrow C_r$	
$U_{r,v}^{c,v} \rightarrow A$	
$C \rightarrow C_r$	if $t \in RF(s)$ then $t \in (U_v^{c,v} \cup U_{r,v}^{c,v} \cup C \cup C_r)$
$C \rightarrow A$	$s \in (IC \cup C)$ or $s \in RC$ and the compensation reservation was successful
$C_r \rightarrow A$	$s \in (IC \cup C)$ or $s \in RC$ and the compensation reservation was successful

Table 2: Subtransaction State Transitions II

Furthermore, there is no way for them to move from a committed state (C_r, C) to an aborted one (A). Therefore, for successful commitment, one of the two subtransactions must first move to a commit state. However, the other subtransaction may next move to an abort state, giving, for example:

$$[..., C_r, ..., A, ...]$$

In this situation, the C_r state is final (i.e. no transitions can be made from it), and the A state is final, so the transaction is in a state of partial commitment indefinitely, a violation of the conditions of correctness. □

Theorem 4.2 *A global transaction T will not be commitable if it contains a subtransaction that is not reservable, that reads from a subtransaction that is not compensatable or preparable, and does not have an explicit commit operator (i.e. $s, t \in T$, $s \notin RES$, $t \notin (COMP \cup P \cup XC)$, $t \in RF(s)$, and $s \neq t$).*

Proof: One can represent the initial state as

$$[..., U, ..., U, ...]$$

where the first U represents the status of the NXC subtransaction t, and the second U represents the status of the non-reservable subtransaction s, that reads from the t. Assuming one does not abort, from the transition definitions, there are only two possible states in which the NXC subtransaction can end up where its values are known:

$$[..., C_r, ..., U, ...]$$

$$[..., C, ..., U, ...]$$

That is, either t can be committed in a recoverable or non-recoverable state. The non-reservable subtransaction cannot proceed until the NXC subtransaction knows its values, so that the non-reservable subtransaction can read them. In the first case, one may not be able to move the state of s to C, since this transition is dependent on the subtransaction execution being successful. Therefore, one is in a case of being in a state of partial commitment indefinitely (since $t \notin COMP$, the status of t cannot move to A). The second case is even worse than the first (since the issue of recoverability is not solved), and the same argument applies. So, one cannot avoid the case where the transaction may end up in a state of partial commitment indefinitely. □

Theorem 4.3 *A global transaction T will not be committable if it contains an $NCPR$ subtransaction that reads from a non-value-preserving reservable subtransaction with an explicit commit operator that is not compensatable or preparable (i.e. $s,t \in T$, $s \in NCPR$, $t \in (((R \cup IR) \cap XC) - (VPR \cup VPIR \cup COMP \cup P))$, $t \in RF(s)$, $s \neq t$).*

Proof: If the reservable subtransaction commits before the $NCPR$ subtransaction, then the $NCPR$ subtransaction may ultimately fail, since its commitment is not guaranteed, and the transaction will be in a state of partial commitment indefinitely. Without actually committing, the reservable subtransaction may, "at best", be able to move to state $U^c_{r,v}$, but cannot move to a state where its values are guaranteed. The $NCPR$ subtransaction can then move to C, or (if it is in XC) $U_{r,v}$. If it moves to C, then the reservable subtransaction may not commit with the same values used by the $NCPR$ subtransaction, so the $NCPR$ subtransaction will remain in a non-recoverable committed state indefinitely. If it moves to $U_{r,v}$ and waits for the reservable subtransaction to commit first so that it can move to C_r, it may end up not being able to commit (since it is not guaranteed). So the transaction would end up in a partially committed state, and would remain there indefinitely. □

Theorem 4.4 *A global transaction T will not be committable if it contains two subtransactions that read from each other that do not have an explicit commit operator, or are not compensatable, preparable, or value-preserving reservable. (i.e. $s,t \in T$, $s,t \in ((NCPR \cup R \cup IR \cup NXC) - ((COMP \cup P \cup VPIR \cup VPR) \cap XC))$, $s \neq t$, $s \in RF(t)$, and $t \in RF(s)$)*

Proof: Again, the initial status of interest could be expressed by:

$$[..., U, ..., U, ...]$$

We will divide the proof into three cases:

(i) $s,t \in NCPR$. The proof of this case follows from the proof of Theorem 4.1.

(ii) $s,t \in NXC$. Neither subtransaction will be to able execute until it has the other subtransaction's values. So, in effect, a deadlock occurs, and the transaction cannot be committed.

(iii) $s \in R \cup IR$, $t \in R \cup IR \cup NCPR$. The proof of this case is covered below.

The status of the subtransactions s and t can never be one where their values are guaranteed (e.g. U^v or $U^{c,v}$) due to their type. Again, for the transaction to commit, one subtransaction must commit before the other; it must go to state C, since it is not recoverable. Suppose after it commits, however, the other subtransaction does commit, but ends up reading different values than the instance from which was read by the subtransaction that committed first. One could end up in such a state:

$$[..., C, ..., C_r, ...]$$

The transaction is in a non-recoverable state, and cannot get out of it, since there is no way to make a transition from the C to C_r state. And, since the subtransactions are not compensatable, one cannot make a transition from C to A, or C_r to A. So condition 5 of the correctness conditions cannot be guaranteed. □

Theorem 4.5 *If a global transaction T has an $NCPR$ subtransaction, then it may not have a reservable subtransaction that is not value-preserving, preparable, or compensatable, that is read by a non-reservable subtransaction (i.e. the following is not allowed $s,t \in T$, $s \in ((R \cup IR) - (VPR \cup VPIR \cup P \cup COMP))$, $t \notin RES$, $s \in RF(t)$, $s \neq t$). Otherwise (if T does not have an $NCPR$ subtransaction) it may not have more than one such subtransaction.*

Proof: For the case where T has an $NCPR$ subtransaction, suppose that there is such a subtransaction as s above. If s commits before the $NCPR$ subtransaction, then the $NCPR$ subtransaction may not commit, and the transaction will be in a state of indefinite partial commitment.

Otherwise, if the $NCPR$ subtransaction commits before s, the following will occur. If the $NCPR$ subtransaction commits before t is guaranteed to commit, then t may not end up committing, and again one is in a state of indefinite partial commitment. For t to be guaranteed to commit, it must either be preparable and execute up to its commit point, or compensatable and commit. However, the values of s will only be guaranteed if s is committed, so t cannot be in a recoverable state. If s ends up committing reading different values than those used by t, t cannot move to the recoverable committed state (C_r). And, t cannot move to the aborted state (A), since the $NCPR$ subtransaction has already committed, and this would leave the transaction in a partially committed state indefinitely.

For the case where T has no $NCPR$ subtransaction, suppose that there are two non-value preserving reservable subtransactions (s_1 and s_2) that are read. The problem that occurs is similar to above. If s_1 is committed before s_2, then s_2 may not commit reading the same values, making the subtransaction that read from it non-recoverable. The subtransaction will not be able to get to a recoverable status, but yet the transaction cannot be aborted, because s_1 has committed and cannot be undone. □

Theorem 4.6 *A transaction T will be committable iff it satisfies all of the following conditions:*

(i) Has at most one subtransaction that is not committable, reservable, or preparable (i.e. $|T \cap NCPR| \leq 1$).

(ii) Has no non-reservable subtransactions that read from a subtransaction that is not compensatable (or preparable) and does not have an explicit commit operator.

(iii) Has no NCPR subtransaction that reads from a non-value-preserving reservable subtransaction with an explicit commit operator that is not compensatable or preparable (i.e. in $(((R \cup IR) \cap XC) - (VPR \cup VPIR \cup COMP \cup P))$).

(iv) Has no two subtransactions that read from each other and that do not have an explicit commit operator, or are not compensatable, preparable, or value preserving reservable. (i.e. subtransactions in $(NXC \cup NCPR \cup R \cup IR)$, and not in $((IC \cup C \cup RC \cup P \cup VPIR \cup VPR) \cap XC)$).

(v) If it has an NCPR subtransaction, then it does not have a non-value preserving reservable subtransaction that is read by a non-reservable subtransaction. Otherwise (if it does not have an NCPR subtransaction), it may not have more than one such subtransaction.

Proof:

(if) To prove this part of the proof, we must show a method for making subtransaction status transitions that will commit the transaction without violating the atomic commitment protocol correctness conditions. Our initial state is:

$$[U, U, ..., U]$$

The first step will be to attempt all the reservations. This will move all subtransactions in $(R \cup IR)$ to state U^c, and all subtransaction in $(VPR \cup VPIR)$ to state $U^{v,c}$. It will also ensure that subtransactions in RC are compensatable. It is possible that some reservations may fail. If a (non-implicit) reservation for a subtransaction in RES fails, then the subtransaction stays in state U and can no longer be considered as reservable. If a reservation for a subtransaction in RC fails, then that subtransaction can no longer be considered as compensatable. If changes to the subtransaction classifications caused by unsuccessful reservations cause the transaction to violate the conditions of this theorem, then the transaction must be aborted, and one ends up in the following state:

$$[A, A, ..., A]$$

If the transaction is still committable, then one will be in the following state:

$$[\overbrace{U, U, ..., U}^{other}, \overbrace{U^c, U^c, ..., U^c}^{(R \cup IR)}, \overbrace{U^{c,v}, U^{c,v}, ..., U^{c,v}}^{(VPR \cup VPIR)}]$$

One can next proceed by:

(1) executing subtransactions in $(P - (RES \cup COMP))$ up to their commit points.

(2) committing compensatable subtransactions in $(COMP - RES)$

(3) executing subtransactions in $(((NCPR \cup RES) \cap XC) \cap RF((P \cup COMP) - RES))$ up to their commit points.

(4) executing subtransaction in $(P \cap RF((P \cup COMP) - RES))$ up to their commit point.

(5) committing subtransactions in $((COMP \cap XC) \cap RF((P \cup COMP) - RES))$.

while preserving read-from dependencies. By Theorem 4.2, all subtransactions that could be in the read-from set of (1) and (2) must be included in (1), (2), (3), (4), and (5). Since the subtransactions in (3), (4), and (5) are in the read-from set of (1) and (2), then any subtransactions in their read-from set must be in the read-from set of subtransactions in (1) and (2). Therefore, the subtransactions in (1), (2), (3), (4), and (5) will only contain subtransactions in their read-from set that are also in (1), (2), (3), (4), and (5). We will consider this step to be phase one, and refer to subtransactions in this phase as PHASE-I subtransactions.

If any of the PHASE-I subtransaction executions above fail, then the transactions should be aborted by moving to:

$$[A, A, ..., A]$$

This will be possible, since the only subtransactions that could possibly be in state C or C_r are those that are also compensatable (in $COMP$). So all subtransactions can make a transition to state A.

If all the executions are successful, then the transaction will be in the following state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((NCPR) \cap XC) \cap RF((P \cup COMP) - RES)$	U_v
$((R \cup IR) \cap XC) \cap RF((P \cup COMP) - RES)$	U_v^c
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$
other (non-PHASE-I)	U

There are now four cases, based on Theorems 4.1 and 4.5, that must be considered.

1. no $NCPR$ subtransaction, and no $(R \cup IR)$ subtransactions in the read-from set of the PHASE-I subtransactions.

2. no $NCPR$ subtransaction, and one $(R \cup IR)$ subtransaction in the read-from set of the PHASE-I subtransactions.

3. one $(NCPR \cap XC)$ subtransaction that is in the read-from set of the PHASE-I subtransactions.

4. one $NCPR$ subtransaction not in (and no $(R \cup IR)$ subtransactions in) the read-from set of the PHASE-I subtransactions.

(Case 1) If there are no $NCPR$ subtransactions and no $(R \cup IR)$ subtransactions that are read from by the PHASE-I subtransactions, then one is in the following state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_u^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_u^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

Since for each of the PHASE-I subtransactions all the subtransactions that are read from know their values and they are also guaranteed, each subtransaction can now move to a recoverable state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_{r,v}^{c,v}$
$(COMP - RES)$	C_r
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_{r,v}^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_{r,v}^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C_r
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

The PHASE-I subtransactions can now all move to the C_r state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	C_r
$(COMP - RES)$	C_r
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	C_r
$P \cap RF((P \cup COMP) - RES)$	C_r
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C_r
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

We will refer to the remaining uncommitted subtransactions as the PHASE-II subtransactions. All of the PHASE-II subtransactions are reservered. We will divide the PHASE-II subtransactions as follows:

Subtransaction Type:	State:
PHASE-I	C_r
$(R \cup IR) \cap NXC$	U^c
$(VPR \cup VPIR) \cap NXC$	$U^{c,v}$
$((R \cup IR) \cap XC) - (P \cup COMP)$	U^c
$((R \cup IR) \cap XC) \cap (COMP)$	U^c
$((R \cup IR) \cap XC) \cap (P \cup COMP)$	U^c
$VPR \cap XC$	$U^{c,v}$

From Theorem 4.4, we know that there must be no cyclical reads between subtransactions in the following class:

$$(NCPR \cup R \cup IR \cup NXC) - ((IC \cup C \cup RC \cup P \cup VPIR \cup VPR) \cap XC)$$

So, one can proceed in the following manner. Execute (and commit) in order of read-from dependencies the subtransactions above that cannot have a read-from cycle. Before executing each subtransaction, for each subtransaction t from which it reads do the following:

- If $t \in (VPR \cap XC)$, move t to state $U_v^{c,v}$.
- If $t \in ((R \cup IR \cup COMP) \cap XC)$, move t to C.
- If $t \in ((R \cup IR \cup P) \cap XC)$, move t to $U_v^{c,v}$.
- If $t \in ((R \cup IR \cup COMP \cup P) \cap XC)$, move t to C.
- If $t \in NXC$ or $t \in ((R \cup IR) \cap XC)$, do nothing. This subtransaction must have already been committed, and should be in state C_r.

After this is done, all NXC and $(R \cup IR \cup XC)$ subtransactions will be in state C_r. Since there might be some subtransactions which did not read from the NXC and $(R \cup IR \cup XC)$ subtransactions, they need to be handled now as follows:

- If $t \in (VPR \cup XC)$, move t to state $U_v^{c,v}$.
- If $t \in (R \cup IR \cup XC \cup COMP)$, move t to C.
- If $t \in (R \cup IR \cup XC \cup P)$, move t to $U_v^{c,v}$.
- If $t \in (R \cup IR \cup XC \cup COMP \cup P)$, move t to C.

Now all subtransactions not in state C_r, are either in state C, or $U_c^{c,v}$. So the subtransactions in C can move to C_r, and the subtransactions in $U_c^{c,v}$ can move to C_r. The transaction has now been successfully committed, since all subtransactions are now in state C_r.

(Case 2) If there are no $NCPR$ subtransactions and one $(R \cup IR)$ subtransaction that is read from by the PHASE-I subtransactions, then one is in the following state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$((R \cup IR) \cap XC) \cap RF((P \cup COMP) - RES)$	U_v^c
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

The only difference in this case from Case 1 is that for the PHASE-I subtransactions, the $(R \cup IR \cup XC)$ subtransaction must be committed first:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$((R \cup IR) \cap XC) \cap RF((P \cup COMP) - RES)$	C_r
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

Now that this subtransaction has been committed, the other PHASE-I subtransactions can proceed as in Case 1. The PHASE-II subtransactions proceed as in Case 1 also.

(Case 3) If there is one $(NCPR \cap XC)$ subtransactions (and no $(R \cup IR)$ subtransaction) that is read from by the PHASE-I subtransactions, then one is in the following state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$(NCPR \cap XC) \cap RF((P \cup COMP) - RES)$	U_v
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

The only difference in this case from Case 1 is that the PHASE-1 $NCPR$ subtransaction must be committed first:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$(NCPR \cap XC) \cap RF((P \cup COMP) - RES)$	C_r
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$

If the $NCPR$ subtransaction commits, the other PHASE-I subtransactions can proceed as in Case 1. The PHASE-II subtransactions proceed as in Case 1 also. If the $NCPR$ subtransaction fails to commit, then the transaction is aborted by aborting all the subtransactions (move to state A).

(Case 4) If there is one $(NCPR)$ subtransaction that is not read from by the PHASE-I subtransactions, then one is in the following state:

Subtransaction Type:	State:
$(P - (RES \cup COMP))$	$U_v^{c,v}$
$(COMP - RES)$	C
$((VPR \cup VPIR) \cap XC) \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$P \cap RF((P \cup COMP) - RES)$	$U_v^{c,v}$
$(COMP \cap XC) \cap RF((P \cup COMP) - RES)$.	C
$R \cup IR$ (non-PHASE-I)	U^c
$VPR \cup VPIR$ (non-PHASE-I)	$U^{c,v}$
$NCPR$	U

From Theorem 4.2 and Theorem 4.3 we know that the $NCPR$ subtransaction can only read from:

- PHASE-I subtransactions

- PHASE-II subtransactions that are in $(P \cup COMP \cup VPR \cup VPIR) \cap XC$

The PHASE-I subtransaction all have known values and guaranteed commitment and values. The PHASE-II subtransactions can be executed as follows:

- $t \in COMP$: move to C.
- $t \in P$: move to $U_v^{c,v}$.
- $t \in VPR \cup VPIR$: move to $U_v^{c,v}$.

Next the $NCPR$ subtransaction is executed. If it fails, the remaining subtransactions can still be undone, so the subtransaction is aborted and all subtransactions move to state A. If the $NCPR$ subtransaction commits, then one can commit the subtransactions from which it read, and proceed to commit the other PHASE-I subtransactions as in Case 1. The PHASE-II subtransactions can be committed the same as in Case 1.

(only if) This part of the theorem is proved by the previous theorems (Theorems 4.1, 4.2, 4.3, 4.4, and 4.5), since the conditions for committability are the negation of the conditions in these (non-committability) theorems. □

5 A Unified Commitment Protocol

Our unified commitment approach deals with committing transactions that have subtransactions with differing commitment methods and semantics. Each transaction will consist of a set of subtransactions where each subtransaction has a set of commitment methods. The set of commitment methods can be empty, or include a combination of prepare, redo, and undo compensation methods. Subtransaction can either have an explicit commit operator, or not have one, which implies that the entire subtransaction must be executed as a whole. Subtransactions without an explicit commit operator must read all values from other subtransactions before beginning, and must write all values to other subtransactions after completing.

Our algorithm follows directly from the section that discusses which transactions are, and are not, committable. We do not assume that the algorithm is given a committable transaction, so the first task is to check to see if the transaction can successfully be committed. If it can, the execution of the transaction can begin, otherwise it must be rejected. One must also check after the reservation phase, as reservation failures could make the transaction uncommittable, although it is possible to have some reservation failures and still be able to commit the transaction.

The following is a description of our unified commitment approach.

Input: Transaction Specification

Output: Transaction Execution

Algorithm:

```
if (the transaction cannot be committed if all reservations succeed) then {
   abort;
}

execute reservations; /* reservations might fail */

if (transaction is still committable) then {
```

```
/* PHASE-I */
while respecting intra-subtransaction dependencies:
- commit compensatable non-reservable subtransactions;
- execute preparable subtransactions (that are not compensatable
    or reservable) up to their prepare points;
- execute subtransactions that have an explicit commit operator and
    are read by the above subtransactions up to their commit points;

if (there is a single subtransaction that is not compensatable,
    preparable, or reservable (NCPR) in the above subtransactions) then {
    execute and commit it;
    if (the single NCPR subtransaction fails) then {
        abort global transaction:
        - undo reservations;
        - compensate committed compensatable subtransactions;
        - exit;
    }
    else {
        commit other subtransactions from above that have not yet been committed;
    }
else if (there is a single non-value preserving reservable subtransaction
    that is not preparable or compensatable) then {
    commit it;
    commit other subtransactions from above that have not yet been committed;
}
else if (there is a single NCPR subtransaction that is not read from
    by the PHASE-I subtransactions) {
    do the following for any subtransaction from which the NCPR subtransaction reads:
    - commit compensatable subtransactions;
    - execute subtransactions with an explicit commitment operator, that
        are not compensatable, up to their commit points.

    execute the NCPR subtransaction;
    if (it fails) then {
        /* Abort the transaction */
        undo successful reservations;
        compensate committed compensatable subtransactions;
        abort any other subtransactions;
    }
    else {
        commit subtransactions from above that have not yet been committed;
    }
}
else {
    commit subtransactions from above that have not yet been committed;
}

/* PHASE-II */
execute and commit subtransactions, that are not value-preserving reservable,
```

```
        compensatable or preparable, or that do not have an explicit commitment
        operator, sequentially in an order compatible with their read-from
        dependencies — before each is executed, execute subtransactions it
        reads from as follows:
    - execute value-preserving reservable subtransactions with an explicit
        commitment operator up to their commit points;
    - commit compensatable subtransactions;
    - execute preparable subtransactions up to their commit points;

    execute and commit any subtransactions that remain uncommitted;
}
else {
    /* Decision to abort global transaction */
    undo successful reservations;
}
```

6 Related Work

In [23] a method for performing the integration of multidatabase systems that use different forms of two phase commitment and three phase commitment is discussed. This topic is beyond the scope of this paper. In this paper three phase commit is not dealt with, and implementations of two phase commitment are assumed to be provided by a visible prepare to commit state.

In [15] a method that uses compensating transactions is presented. The method provides a relaxation of standard atomicity called *semantic atomicity*. This method uses a new transaction/correctness model. In particular, isolation of the ACID properties will not be provided.

In [2] an overview of multidatabase transaction management is presented. While [2] does survey atomic commitment work and does briefly describe a unified algorithm, our work differs in the following ways:

- it considers the effect of not having an explicit commit operation, i.e. the entire subtransaction must be submitted at once.
- it considers the semantics of the subtransactions.
- it considers redoable subtransactions that require an explicit reservation (or pre-step).
- it considers compensatable subtransactions that require an explicit reservation (or pre-step).
- it includes subtransactions for which multiple commitment approaches exist.
- we formally show the types of transactions that are, and are not, committable.

In [16] a method that combines compensation with a retry approach is presented. This approach appears to be less general than the approach in [2], which is discussed above.

In [18] an approach is presented that combines retriable and compensatable subtransactions, as well as, a possible "pivot" subtransaction that may be neither retriable or compensatable. This approach is basically the same as that described in [2] (which is discussed above).

7 Conclusions

We have presented a taxonomy of multidatabase atomic commitment protocols based on the basic approach (prepare, redo, or undo) and specific implementation strategy that are used. All of the specific implementa-

tion strategies use either autonomy violation or transaction class restriction, or a combination of these two approaches.

A taxonomy of multidatabase global transaction subtransactions was presented. This taxonomy was based on the commitment approaches in the protocol taxonomy, transaction semantics, and the operational interface provided by the component system. We have identified transaction conditions where atomic commitment is and is not possible based on this subtransaction taxonomy.

Finally, we have presented a unified approach to atomic commitment that is based on the theory developed to show the class of committable global transactions. Our unified protocol is more general than any of the other approaches with which we are familiar. We are currently implementing a unified commitment approach based on this work in our prototype multidatabase system InterBase-Star, the successor to the InterBase Multidatabase System [8, 6, 5].

References

[1] K. Barker and M. Özsu. Reliable transaction execution in multidatabase systems. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 344–347, Kobe, Japan, Apr. 1991.

[2] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, 1(2):181–239, October 1992.

[3] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 215–224, May 1990.

[4] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Transaction managemnet issues in a failure-prone multidatabase system environment. *VLDB Journal*, 1(1):1–39, July 1992.

[5] O. Bukhres, J. Chen, W. Du, A. Elmagarmid, and R. Pezzoli. InterBase : An Execution Environment for Global Applications over Distributed, Heterogeneous, and Autonomous Software Systems. *IEEE Computer*, 1993. (to appear).

[6] J. Chen, O. A. Bukhres, and A. K. Elmagarmid. IPL: A Multidatabase Transaction Specification Language. In *Proc. of the 13th International Conference on Distributed Computing Systems*, 1993. (to appear).

[7] A. Elmagarmid, J. Jing, and W. Kim. Global committability in multidatabase systems. Technical Report CSD-TR-91-017, Department of Computer Science, Purdue University, 1991.

[8] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 507–581, Brisbane, Australia, Aug. 1990.

[9] A. K. Elmagarmid, J. Jing, J. G. Mullen, and J. Sharif-Askary. Reservable transactions: An approach for reliable multidatabase transaction management. Technical Report CSD-TR 92-012, Department of Computer Science, Purdue University, March 1992.

[10] D. Georgakopoulos. Multidatabase recoverability and recovery. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 348–355, Kobe, Japan, Apr. 1991.

[11] J. N. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, pages 624–633. Springer-Verlag, Berlin, 1978.

[12] M. Hsu and A. Silberschatz. Unilateral commit: A new paradigm for reliable distributed transaction processing. In *Proceedings of the 7th International Conference on Data Engineering*, pages 296–304, Kobe, Japan, Apr. 1991.

[13] H. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, Aug. 1990.

[14] B. Lampson and H. Sturgis. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox Palo Alto Research Center, 1976.

[15] E. Levy, H. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Denver, Colorado, May 1991.

[16] E. Levy, H. Korth, and A. Silberschatz. A theory of relaxed atomicity. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Aug. 1991.

[17] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. Ensuring transaction atomicity in multidatabase systems. Technical Report TR-92-12, University of Texas at Austin Department of Computer Science, 1992.

[18] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for multidatabase systems. In *Proceedings of International Conference on Distributed Computing Systems*, June 1992.

[19] J. G. Mullen, A. K. Elmagarmid, W. Kim, and J. Sharif-Askary. On the impossibility of atomic commitment in multidatabase systems. In *Proceedings Of The International Conference on Systems Integration*, pages 625–634, Morristown, New Jersey, June 1992.

[20] P. Muth and T. Rakow. Atomic commitment for integrated database systems. In *Proceedings of the 7th Intl. Conf. on Data Engineering*, pages 296–304, Kobe, Japan, Apr. 1991.

[21] W. Perrizo, J. Rajkumar, and P. Ram. HYDRO: a heterogeneous distributed database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 32–39, Denver, Colorado, USA, May 1991.

[22] N. Soparkar, H. F. Korth, and A. Siberschatz. Failure-resilient transaction management in multidatabases. *IEEE Computer*, 24(12):28–36, December 1991.

[23] A. Tal and R. Alonso. Integration of commit protocols in heterogeneous databases. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 27–34, Baltimore, Maryland, USA, November 1992.

[24] A. Wolski and J. Veijalainen. 2PC Agent method: Achieving serializability in presence of failures in a heterogeneous multidatabase. In *Proceedings of PARBASE-90*, Miami Beach, Florida, 1990.